**Name : Tanishq Thuse**

**Branch : CS(AI)**

**Year : SY**

**Div : B**

**Roll no. : 60**

**PRN : 12310237**

**Title : OS Assignment-5 Implementation of Banking Algorithm**

**Q1) Safety Algorithm**

**Code :**

#include <iostream>

using namespace std;

int main()

{

int n, m, i, j, k;

// Input the number of processes and resources

cout << "Enter the number of processes: ";

cin >> n;

cout << "Enter the number of resources: ";

cin >> m;

// Declare matrices and arrays

int alloc[n][m], max[n][m], avail[m], f[n], ans[n], ind = 0;

// Input Allocation Matrix

cout << "Enter the allocation matrix (row by row):\n";

for (i = 0; i < n; i++) {

for (j = 0; j < m; j++) {

cin >> alloc[i][j];

}

}

// Input Maximum Matrix

cout << "Enter the maximum matrix (row by row):\n";

for (i = 0; i < n; i++) {

for (j = 0; j < m; j++) {

cin >> max[i][j];

}

}

// Input Available Resources

cout << "Enter the available resources:\n";

for (i = 0; i < m; i++) {

cin >> avail[i];

}

// Initialize flags array

for (k = 0; k < n; k++) {

f[k] = 0;

}

// Calculate the need matrix

int need[n][m];

for (i = 0; i < n; i++) {

for (j = 0; j < m; j++) {

need[i][j] = max[i][j] - alloc[i][j];

}

}

// Banker's Algorithm to find the safe sequence

int y = 0;

for (k = 0; k < n; k++) {

for (i = 0; i < n; i++) {

if (f[i] == 0) {

int flag = 0;

for (j = 0; j < m; j++) {

if (need[i][j] > avail[j]){

flag = 1;

break;

}

}

if (flag == 0) {

ans[ind++] = i;

for (y = 0; y < m; y++) {

avail[y] += alloc[i][y];

}

f[i] = 1;

}

}

}

}

// Check if the system is in a safe state

int flag = 1;

for (i = 0; i < n; i++) {

if (f[i] == 0) {

flag = 0;

cout << "The given sequence is not safe\n";

break;

}

}

// Print the safe sequence if it exists

if (flag == 1) {

cout << "Following is the SAFE Sequence:\n";

for (i = 0; i < n - 1; i++) {

cout << " P" << ans[i] << " ->";

}

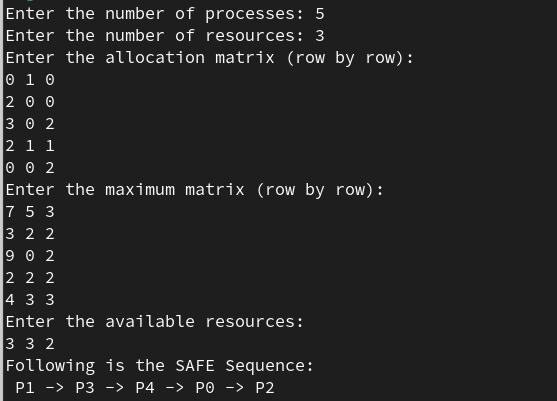
cout << " P" << ans[n - 1] << endl;

}

return 0;

}

**Output :**



**Q2) Resource Request/Allocation Graph**

**Code :**

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

class ResourceAllocationGraph {

private:

int numProcesses;

int numResources;

vector<vector<int>> allocation; // Allocation matrix

vector<vector<int>> request; // Request matrix

vector<int> available; // Available resources

vector<vector<int>> maxDemand; // Maximum demand for resources

public:

ResourceAllocationGraph(int processes, int resources) {

numProcesses = processes;

numResources = resources;

allocation.resize(processes, vector<int>(resources, 0));

request.resize(processes, vector<int>(resources, 0));

available.resize(resources, 0);

maxDemand.resize(processes, vector<int>(resources, 0));

}

void setAvailableResources() {

cout << "Enter available resources for " << numResources << " resources: ";

for (int i = 0; i < numResources; i++) {

cin >> available[i];

}

}

void setMaxDemand() {

cout << "Enter the maximum demand for each process and each resource:\n";

for (int i = 0; i < numProcesses; i++) {

cout << "Process P" << i << ": ";

for (int j = 0; j < numResources; j++) {

cin >> maxDemand[i][j];

}

}

}

void requestResources() {

for (int i = 0; i < numProcesses; i++) {

cout << "Enter the resource request for process P" << i << ": ";

for (int j = 0; j < numResources; j++) {

cin >> request[i][j];

}

}

}

void allocateResources() {

for (int i = 0; i < numProcesses; i++) {

for (int j = 0; j < numResources; j++) {

allocation[i][j] += request[i][j];

available[j] -= request[i][j];

request[i][j] = 0;

}

}

}

bool isDeadlock() {

vector<bool> finish(numProcesses, false);

vector<int> work = available;

queue<int> q;

for (int i = 0; i < numProcesses; i++) {

if (!finish[i]) {

bool canProceed = true;

for (int j = 0; j < numResources; j++) {

if (request[i][j] > work[j]) {

canProceed = false;

break;

}

}

if (canProceed) {

finish[i] = true;

q.push(i);

for (int j = 0; j < numResources; j++) {

work[j] += allocation[i][j];

}

}

}

}

// Check if there is any process that could not finish

for (int i = 0; i < numProcesses; i++) {

if (!finish[i]) {

cout << "Deadlock detected involving process P" << i << endl;

return true;

}

}

cout << "No deadlock detected." << endl;

return false;

}

};

int main() {

int processes, resources;

cout << "Enter the number of processes: ";

cin >> processes;

cout << "Enter the number of resources: ";

cin >> resources;

ResourceAllocationGraph rag(processes, resources);

// Set available resources

rag.setAvailableResources();

// Set maximum demand for each process

rag.setMaxDemand();

// Request resources for each process

rag.requestResources();

// Allocate resources to processes

rag.allocateResources();

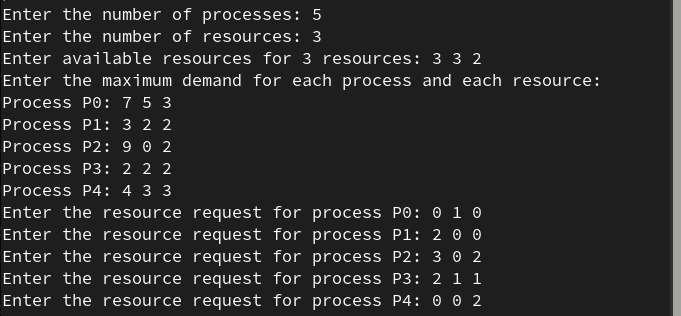
// Check for deadlock

rag.isDeadlock();

return 0;

}

**Output :**

****